5.3 Escalonamento FCFS (*First-Come, First Served*)

A forma de escalonamento mais elementar consiste em simplesmente atender as tarefas em sequência, à medida em que elas se tornam prontas (ou seja, conforme sua ordem de chegada na fila de tarefas prontas). Esse algoritmo é conhecido como FCFS – *First Come - First Served* – e tem como principal vantagem sua simplicidade.

Para dar um exemplo do funcionamento do algoritmo FCFS, consideremos as tarefas na fila de tarefas prontas, com suas durações previstas de processamento e datas de ingresso no sistema, descritas na tabela a seguir:

tarefa	t_1	t_2	t_3	t_4
ingresso	0	0	1	3
duração	5	2	4	3

O diagrama da figura 13 mostra o escalonamento do processador usando o algoritmo FCFS cooperativo (ou seja, sem *quantum* ou outras interrupções). Os quadros sombreados representam o uso do processador (observe que em cada instante apenas uma tarefa ocupa o processador). Os quadros brancos representam as tarefas que já ingressaram no sistema e estão aguardando o processador (tarefas prontas).

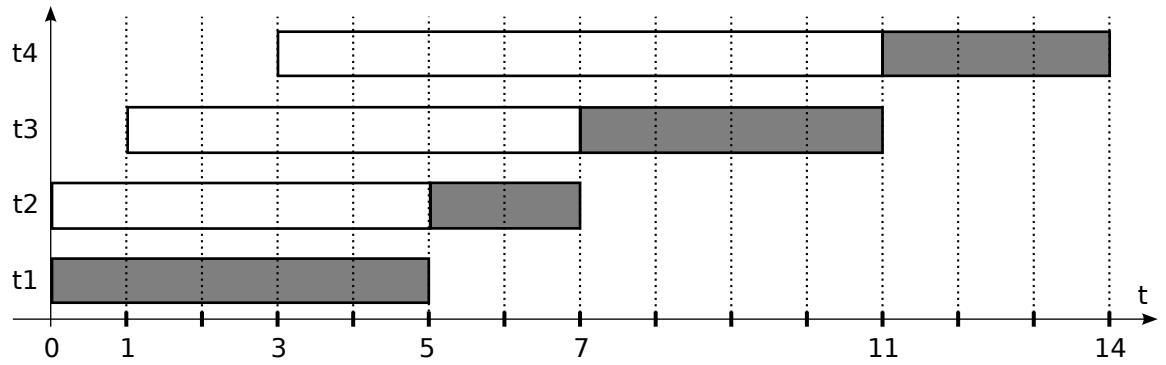

Figura 13: Escalonamento FCFS.

Calculando o tempo médio de execução (T_t, a média de $t_t(t_i)$) e o tempo médio de espera (T_w, a média de $t_w(t_i)$) para o algoritmo FCFS, temos:

$$\begin{aligned} T_t &= \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(5-0) + (7-0) + (11-1) + (14-3)}{4} \\ &= \frac{5+7+10+11}{4} = \frac{33}{4} = 8.25s \end{aligned}$$

$$\begin{aligned} T_w &= \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(0-0) + (5-0) + (7-1) + (11-3)}{4} \\ &= \frac{0+5+6+8}{4} = \frac{19}{4} = 4.75s \end{aligned}$$

O escalonamento FCFS não leva em conta a importância das tarefas nem seu comportamento em relação aos recursos. Por exemplo, com esse algoritmo as tarefas

orientadas a entrada/saída irão receber menos tempo de processador que as tarefas orientadas a processamento (pois geralmente não usam integralmente seus *quanta* de tempo), o que pode ser prejudicial para aplicações interativas.

A adição da preempção por tempo ao escalonamento FCFS dá origem a outro algoritmo de escalonamento bastante popular, conhecido como **escalonamento por revezamento**, ou *Round-Robin*. Considerando as tarefas definidas na tabela anterior e um quantum $t_q = 2s$, seria obtida a sequência de escalonamento apresentada na figura 14.

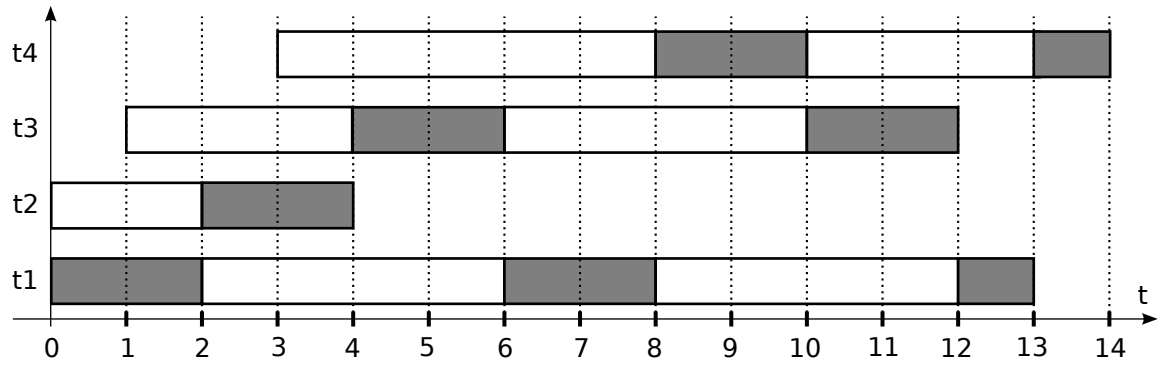

Figura 14: Escalonamento *Round-Robin*.

Na figura 14, é importante observar que a execução das tarefas não obedece uma sequência óbvia como $t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_1 \rightarrow t_2 \rightarrow \ldots$, mas uma sequência bem mais complexa: $t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_1 \rightarrow t_4 \rightarrow t_3 \rightarrow \ldots$. Isso ocorre por causa da ordem das tarefas na fila de tarefas prontas. Por exemplo, a tarefa t_1 para de executar e volta à fila de tarefas prontas no instante $t = 2$, antes de t_4 ter entrado no sistema (em $t = 3$). Por isso, t_1 retorna ao processador antes de t_4 (em $t = 6$). A figura 15 detalha a evolução da fila de tarefas prontas ao longo do tempo, para esse exemplo.

Calculando o tempo médio de execução T_t e o tempo médio de espera T_w para o algoritmo *round-robin*, temos:

$$\begin{aligned} T_t &= \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(13-0) + (4-0) + (12-1) + (14-3)}{4} \\ &= \frac{13+4+11+11}{4} = \frac{39}{4} = 9.75s \end{aligned}$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{8+2+7+8}{4} = \frac{25}{4} = 6.25s$$

Observa-se o aumento nos tempos T_t e T_w em relação ao algoritmo FCFS simples, o que mostra que o algoritmo *round-robin* é menos eficiente para a execução de tarefas em lote. Entretanto, por distribuir melhor o uso do processador entre as tarefas ao longo do tempo, ele pode proporcionar tempos de resposta bem melhores às aplicações interativas.

O aumento da quantidade de trocas de contexto também tem um impacto negativo na eficiência do sistema operacional. Quanto menor o número de trocas de contexto e menor a duração de cada troca, mais tempo sobrará para a execução das tarefas em

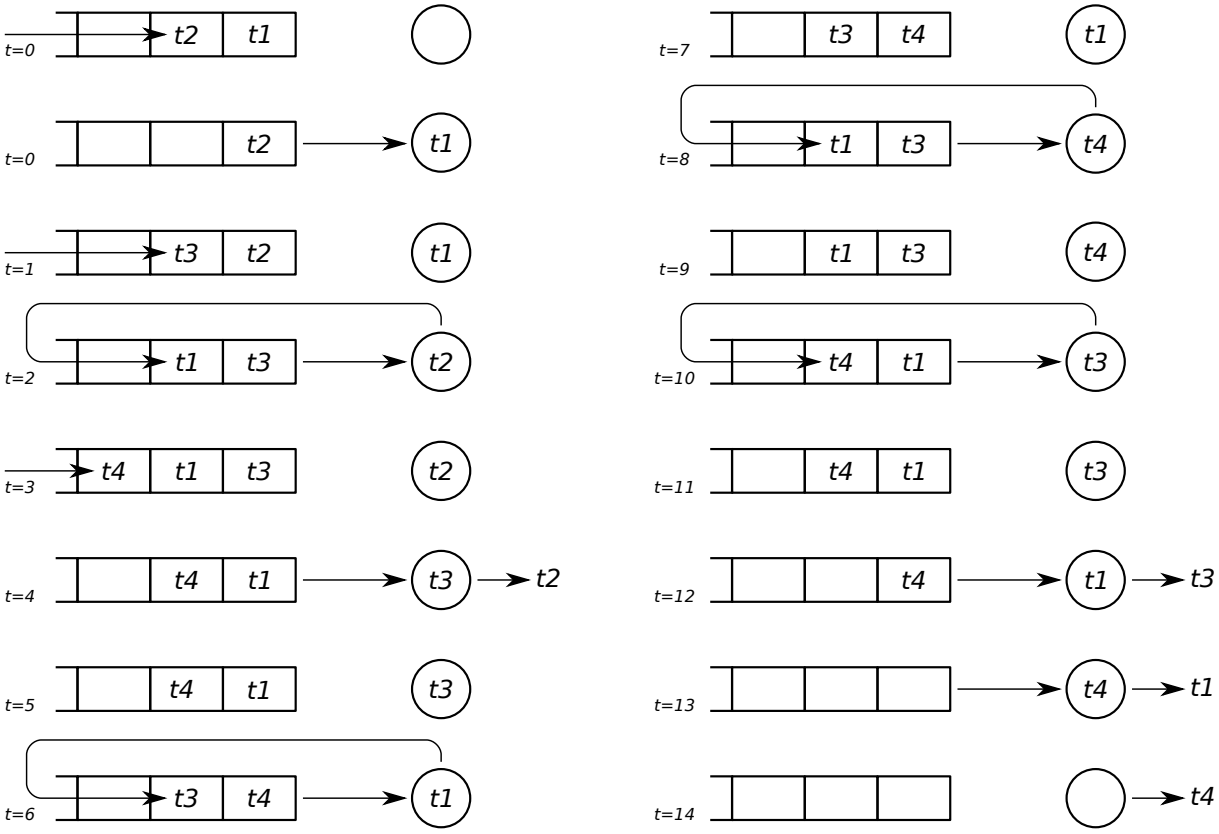

Figura 15: Evolução da fila de tarefas prontas no escalonamento *Round-Robin*.

si. Assim, é possível definir uma **medida de eficiência** $\mathcal{E}$ do uso do processador, em função das durações médias do *quantum* de tempo t_q e da troca de contexto t_{tc}:

$$\mathcal{E} = \frac{t_q}{t_q + t_{tc}}$$

Por exemplo, um sistema no qual as trocas de contexto duram $1ms$ e cujo *quantum* médio é de $20ms$ terá uma eficiência $\mathcal{E} = \frac{20}{20+1} = 95,2\%$. Caso a duração do quantum seja reduzida para $2ms$, a eficiência cairá para $\mathcal{E} = \frac{2}{2+1} = 66,7\%$. A eficiência final da gerência de tarefas é influenciada por vários fatores, como a carga do sistema (mais tarefas ativas implicam em mais tempo gasto pelo escalonador, aumentando t_{tc}) e o perfil das aplicações (aplicações que fazem muita entrada/saída saem do processador antes do final de seu *quantum*, diminuindo o valor médio de t_q).

5.4 Escalonamento SJF (*Shortest Job First*)

O algoritmo de escalonamento que proporciona os menores tempos médios de execução e de espera é conhecido como *menor tarefa primeiro*, ou SJF (*Shortest Job First*). Como o nome indica, ele consiste em atribuir o processador à menor (mais curta) tarefa da fila de tarefas prontas. Pode ser provado matematicamente que esta estratégia sempre proporciona os menores tempos médios de espera. Aplicando-se este algoritmo às tarefas da tabela anterior, obtém-se o escalonamento apresentado na figura 16.

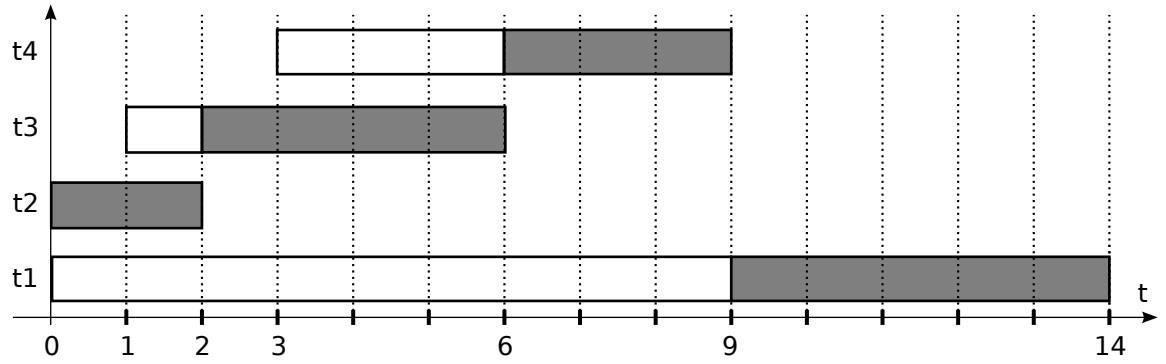

Figura 16: Escalonamento SJF.

Calculando o tempo médio de execução T_t e o tempo médio de espera T_w para o algoritmo SJF, temos:

$$\begin{aligned} T_t &= \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(14-0) + (2-0) + (6-1) + (9-3)}{4} \\ &= \frac{14+2+5+6}{4} = \frac{27}{4} = 6.75s \\ T_w &= \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(9-0) + (0-0) + (2-1) + (6-3)}{4} \\ &= \frac{9+0+1+3}{4} = \frac{13}{4} = 3.25s \end{aligned}$$

Deve-se observar que o comportamento expresso na figura 16 corresponde à versão cooperativa do algoritmo SJF: o escalonador aguarda a conclusão de cada tarefa para decidir quem irá receber o processador. No caso preemptivo, o escalonador deve comparar a duração prevista de cada nova tarefa que ingressa no sistema com o tempo restante de processamento das demais tarefas presentes, inclusive aquela que está executando no momento. Essa abordagem é denominada por alguns autores de *menor tempo restante primeiro* (SRTF – *Short Remaining Time First*) [Tanenbaum, 2003].

A maior dificuldade no uso do algoritmo SJF consiste em estimar a priori a duração de cada tarefa, ou seja, antes de sua execução. Com exceção de algumas tarefas em lote ou de tempo real, essa estimativa é inviável; por exemplo, como estimar por quanto tempo um editor de textos irá ser utilizado? Por causa desse problema, o algoritmo SJF puro é pouco utilizado. No entanto, ao associarmos o algoritmo SJF à preempção por tempo, esse algoritmo pode ser de grande valia, sobretudo para tarefas orientadas a entrada/saída.

Suponha uma tarefa orientada a entrada/saída em um sistema preemptivo com $t_q = 10ms$. Nas últimas 3 vezes em que recebeu o processador, essa tarefa utilizou $3ms$, $4ms$ e $4.5ms$ de cada quantum recebido. Com base nesses dados históricos, é possível estimar qual a duração da execução da tarefa na próxima vez em que receber o processador. Essa estimativa pode ser feita por média simples (cálculo mais rápido) ou por extrapolação (cálculo mais complexo, podendo influenciar o tempo de troca de contexto t_{tc}).

A estimativa de uso do próximo quantum assim obtida pode ser usada como base para a aplicação do algoritmo SJF, o que irá priorizar as tarefas orientadas a entrada/saída, que usam menos o processador. Obviamente, uma tarefa pode mudar de comportamento repentinamente, passando de uma fase de entrada/saída para uma fase de processamento, ou vice-versa. Nesse caso, a estimativa de uso do próximo *quantum* será incorreta durante alguns ciclos, mas logo voltará a refletir o comportamento atual da tarefa. Por essa razão, apenas a história recente da tarefa deve ser considerada (3 a 5 últimas ativações).

Outro problema associado ao escalonamento SJF é a possibilidade de *inanição* (*starvation*) das tarefas mais longas. Caso o fluxo de tarefas curtas chegando ao sistema seja elevado, as tarefas mais longas nunca serão escolhidas para receber o processador e vão literalmente "morrer de fome", esperando na fila sem poder executar. Esse problema pode ser resolvido através de técnicas de envelhecimento de tarefas, como a apresentada na Seção 5.5.2.

5.5 Escalonamento por prioridades

Vários critérios podem ser usados para ordenar a fila de tarefas prontas e escolher a próxima tarefa a executar; a data de ingresso da tarefa (usada no FCFS) e sua duração prevista (usada no SJF) são apenas dois deles. Inúmeros outros critérios podem ser especificados, como o comportamento da tarefa (em lote, interativa ou de tempo-real), seu proprietário (administrador, gerente, estagiário), seu grau de interatividade, etc.

No escalonamento por prioridades, a cada tarefa é associada uma prioridade, geralmente na forma de um número inteiro. Os valores de prioridade são então usados para escolher a próxima tarefa a receber o processador, a cada troca de contexto. O algoritmo de escalonamento por prioridades define um modelo genérico de escalonamento, que permite modelar várias abordagens, entre as quais o FCFS e o SJF.

Para ilustrar o funcionamento do escalonamento por prioridades, serão usadas as tarefas descritas na tabela a seguir, que usam uma escala de prioridades positiva (ou seja, onde valores maiores indicam uma prioridade maior):

tarefa	t_1	t_2	t_3	t_4
ingresso	0	0	1	3
duração	5	2	4	3
prioridade	2	3	1	4

O diagrama da figura 17 mostra o escalonamento do processador usando o algoritmo por prioridades em modo cooperativo (ou seja, sem *quantum* ou outras interrupções).

Calculando o tempo médio de execução T_t e o tempo médio de espera T_w para esse algoritmo, temos:

$$\begin{aligned} T_t &= \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(7-0) + (2-0) + (14-1) + (10-3)}{4} \\ &= \frac{7+2+13+7}{4} = \frac{29}{4} = 7.25s \end{aligned}$$

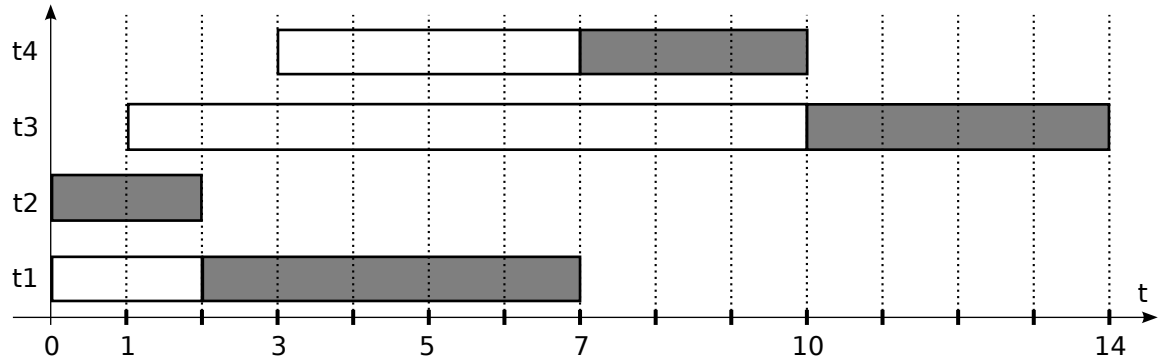

Figura 17: Escalonamento por prioridades (cooperativo).

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{(2-0) + (0-0) + (10-1) + (7-3)}{4}$$
$$= \frac{2+0+9+4}{4} = \frac{15}{4} = 3.75s$$

Quando uma tarefa de maior prioridade se torna disponível para execução, o escalonador pode decidir entregar o processador a ela, trazendo a tarefa atual de volta para a fila de prontas. Nesse caso, temos um escalonamento por prioridades *preemptivo*, cujo comportamento é apresentado na figura 18 (observe que, quando t_4 ingressa no sistema, ela recebe o processador e t_1 volta a esperar na fila de prontas).

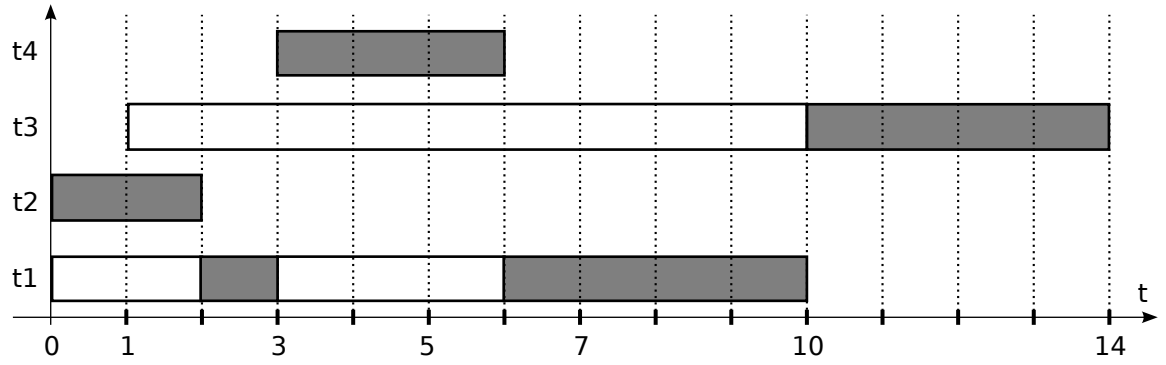

Figura 18: Escalonamento por prioridades (preemptivo).

Calculando o tempo médio de execução T_t e o tempo médio de espera T_w para esse algoritmo, temos:

$$T_t = \frac{t_t(t_1) + t_t(t_2) + t_t(t_3) + t_t(t_4)}{4} = \frac{(10-0) + (2-0) + (14-1) + (6-3)}{4}$$
$$= \frac{10+2+13+3}{4} = \frac{28}{4} = 7s$$

$$T_w = \frac{t_w(t_1) + t_w(t_2) + t_w(t_3) + t_w(t_4)}{4} = \frac{5+0+9+0}{4} = \frac{14}{4} = 3.5s$$